

Ground Systems Development Environment (GSDE) Software Configuration Management

Victor E. Church

D. Long

Ray Hartenstein

Computer Sciences Corporation

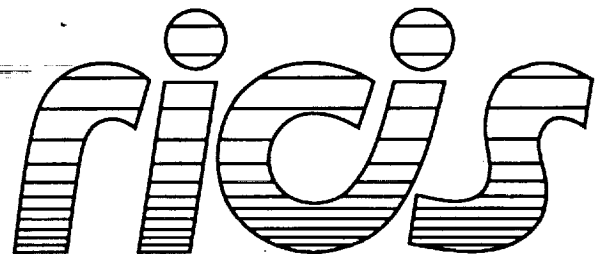
Alfredo Perez-Davila

University of Houston-Clear Lake

April 30, 1992

Cooperative Agreement NCC 9-16
Research Activity No. SE.34

NASA Johnson Space Center
Mission Operations Directorate
Space Station Ground Systems Division



Research Institute for Computing and Information Systems
University of Houston-Clear Lake

TECHNICAL REPORT

The RICIS Concept

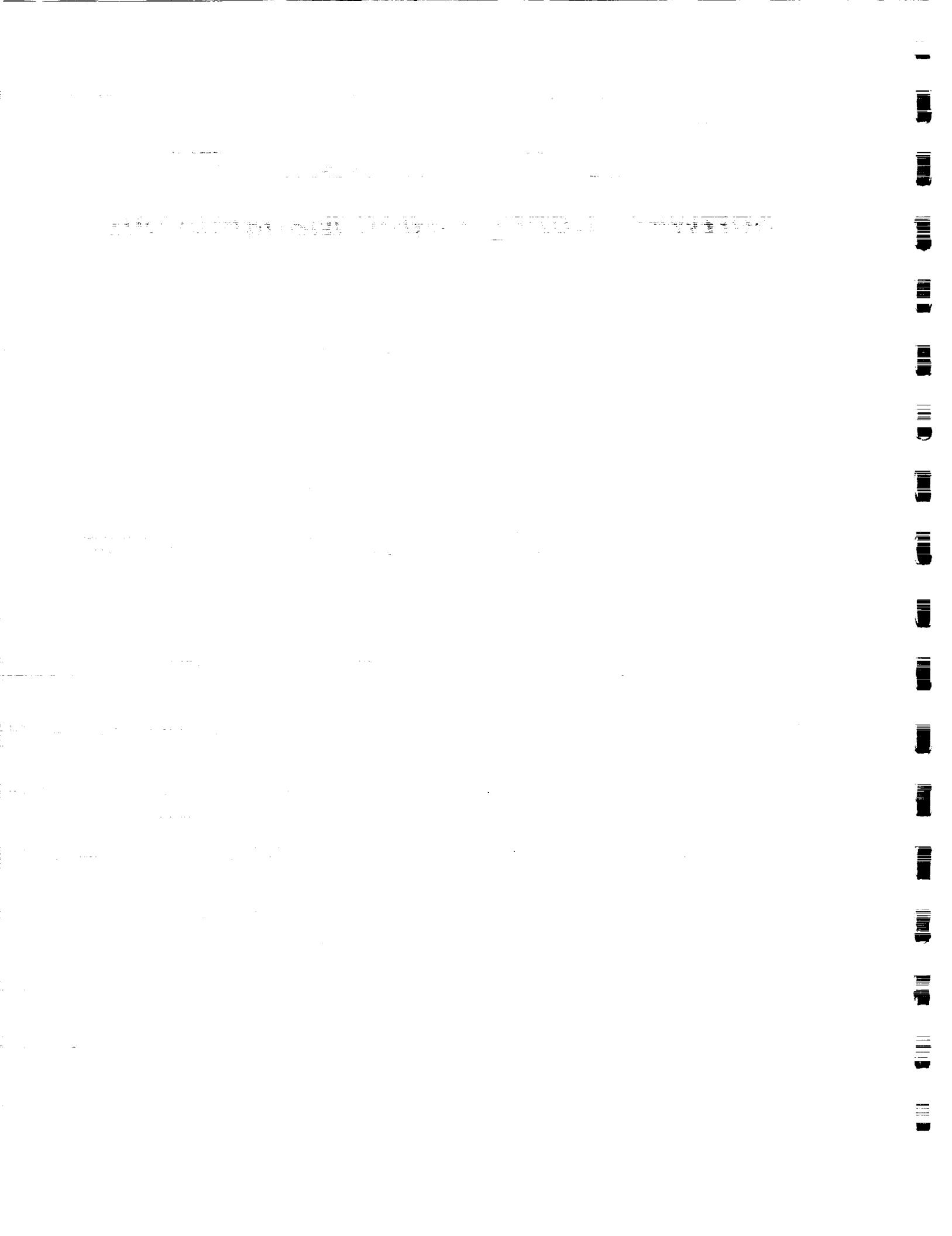
The University of Houston-Clear Lake established the Research Institute for Computing and Information Systems (RICIS) in 1986 to encourage the NASA Johnson Space Center (JSC) and local industry to actively support research in the computing and information sciences. As part of this endeavor, UHCL proposed a partnership with JSC to jointly define and manage an integrated program of research in advanced data processing technology needed for JSC's main missions, including administrative, engineering and science responsibilities. JSC agreed and entered into a continuing cooperative agreement with UHCL beginning in May 1986, to jointly plan and execute such research through RICIS. Additionally, under Cooperative Agreement NCC 9-16, computing and educational facilities are shared by the two institutions to conduct the research.

The UHCL/RICIS mission is to conduct, coordinate, and disseminate research and professional level education in computing and information systems to serve the needs of the government, industry, community and academia. RICIS combines resources of UHCL and its gateway affiliates to research and develop materials, prototypes and publications on topics of mutual interest to its sponsors and researchers. Within UHCL, the mission is being implemented through interdisciplinary involvement of faculty and students from each of the four schools: Business and Public Administration, Education, Human Sciences and Humanities, and Natural and Applied Sciences. RICIS also collaborates with industry in a companion program. This program is focused on serving the research and advanced development needs of industry.

Moreover, UHCL established relationships with other universities and research organizations, having common research interests, to provide additional sources of expertise to conduct needed research. For example, UHCL has entered into a special partnership with Texas A&M University to help oversee RICIS research and education programs, while other research organizations are involved via the "gateway" concept.

A major role of RICIS then is to find the best match of sponsors, researchers and research objectives to advance knowledge in the computing and information sciences. RICIS, working jointly with its sponsors, advises on research needs, recommends principals for conducting the research, provides technical and administrative support to coordinate the research and integrates technical results into the goals of UHCL, NASA/JSC and industry.

***Ground Systems Development
Environment (GSDE)
Software Configuration Management***



RICIS Preface

This research was conducted under auspices of the Research Institute for Computing and Information Systems by Computer Sciences Corporation in cooperation with the University of Houston-Clear Lake. The members of the research team were: Victor E. Church, D. Long and Ray Hartenstein from CSC and Alfredo Perez-Davila from UHCL. Mr. Robert E. Coady was CSC program manager for this project during the initial phase. Later, Mr. Ray Hartenstein assumed the role of CSC program manager. Dr. Perez-Davila also served as RICIS research coordinator.

Funding was provided by the Mission Operations Directorate, NASA/JSC through Cooperative Agreement NCC 9-16 between the NASA Johnson Space Center and the University of Houston-Clear Lake. The NASA research coordinator for this activity was Thomas G. Price of the ADPE and Support Systems Office, Space Station Ground Systems Division, Mission Operations Directorate, NASA/JSC.

The views and conclusions contained in this report are those of the authors and should not be interpreted as representative of the official policies, either express or implied, of UHCL, RICIS, NASA or the United States Government.

1. The first part of the document is a letter from the President of the United States to the Congress, dated January 3, 1862. It is a very important document, as it contains the President's annual message to Congress, which is a key document in the history of the United States.

2. The second part of the document is a report from the Secretary of the Treasury, dated January 3, 1862. It is a very important document, as it contains the Secretary's annual report to Congress, which is a key document in the history of the United States.

3. The third part of the document is a report from the Secretary of the Interior, dated January 3, 1862. It is a very important document, as it contains the Secretary's annual report to Congress, which is a key document in the history of the United States.

4. The fourth part of the document is a report from the Secretary of the War, dated January 3, 1862. It is a very important document, as it contains the Secretary's annual report to Congress, which is a key document in the history of the United States.

5. The fifth part of the document is a report from the Secretary of the Navy, dated January 3, 1862. It is a very important document, as it contains the Secretary's annual report to Congress, which is a key document in the history of the United States.

6. The sixth part of the document is a report from the Secretary of the State, dated January 3, 1862. It is a very important document, as it contains the Secretary's annual report to Congress, which is a key document in the history of the United States.

7. The seventh part of the document is a report from the Secretary of the Army, dated January 3, 1862. It is a very important document, as it contains the Secretary's annual report to Congress, which is a key document in the history of the United States.

8.

9. The eighth part of the document is a report from the Secretary of the Navy, dated January 3, 1862. It is a very important document, as it contains the Secretary's annual report to Congress, which is a key document in the history of the United States.

10. The ninth part of the document is a report from the Secretary of the Army, dated January 3, 1862. It is a very important document, as it contains the Secretary's annual report to Congress, which is a key document in the history of the United States.

11.

12.

13.

14.

15.

16.

Ground Systems Development Environment (GSDE) Software Configuration Management

Prepared for

Lyndon B. Johnson Space Center
National Aeronautics and Space Administration
Houston, Texas

by

Computer Sciences Corporation
System Sciences Division
Beltsville, Maryland and League City, Texas

and

The University of Houston - Clear Lake
Research Institute for Computers and Information Sciences
Clear Lake, Texas

under

Subcontract No. 075
RICIS Research Activity No. SE-34
NASA Cooperative Agreement NCC 9-16

April 1992

Preparation:	V. Church
Quality Assurance:	D. Long
Approval:	R. Hartenstein
	A. Perez-Davila

Abstract

This report presents a review of the software configuration management (CM) plans developed for the Space Station Training Facility (SSTF) and the Space Station Control Center. The scope of the CM assessed in this report is the Systems Integration and Testing Phase of the Ground Systems development life cycle. This is the period following coding and unit test and preceding delivery to operational use. This report is one of a series from a study of the interfaces among the Ground Systems Development Environment (GSDE), the development systems for the SSTF and the SSCC, and the target systems for SSCC and SSTF. This is the last report in the series.

The focus of this report is on the CM plans developed by the contractors for the Mission Systems Contract (MSC) and the Training Systems Contract (TSC). CM requirements are summarized and described in terms of operational software development. The software workflows proposed in the TSC and MSC plans are reviewed in this context, and evaluated against the CM requirements defined in earlier study reports. Recommendations are made to improve the effectiveness of CM while minimizing its impact on the developers.

PRECEDING PAGE BLANK NOT FILMED

Table of Contents

Section 1 - Introduction	1
1.1 Assessment summary.....	1
1.2 Ground system software development.....	2
1.3 Organization of Report	3
1.4 References and related documentation	4
Section 2 - Requirements (Analysis Context)	5
2.1 Goals of CM requirements.....	5
2.2 CM Requirements Summary.....	6
2.3 Space Station SSE CM support.....	8
2.4 Options and alternatives.....	9
Section 3 - Operational Considerations	11
3.1 Development process overview.....	12
3.2 Clean-slate approach to formal CM	14
3.2.1 IQTE-resident product files.....	18
3.2.2 GS/SPF-resident product files	19
3.3 Combined formal/informal CM	20
3.4 Using the SPE as a conduit.....	22
3.5 The mosaic effect	24
3.5.1 Compilation libraries.....	24
3.5.2 Testbed construction	25
Section 4 - Space Station Control Center	27
4.1 Overall software workflow	27
4.2 Configuration management.....	28
4.3 Assessment.....	28
Section 5 - Space Station Training Facility	31
5.1 Overall software workflow	31
5.2 Configuration management.....	31
5.3 Assessment.....	32
Section 6 - Recommendations.....	35
6.1 Space Station Control Center.....	35

PRECEDING PAGE BLANK NOT FILMED

6.2 Space Station Training Facility	36
Glossary	37

List of Figures

1. Ground Software Development Environment	3
2. Develop-build-test process.	12
3. CM Process symbols	13
4. Clean-slate approach to formal CM	16
5. IQTE-resident product files	18
6. Storing product files in the GS/SPF	19
7. Combining formal and informal CM	21
8. Using the SPE as a conduit	23
9. Mixed libraries in the IQTE	24
10. Mixed product files in the IQTE	26

Section 1 - Introduction

As part of the Space Station Freedom Program, the Mission Operations Directorate (MOD) at JSC¹ is developing a Space Station Training Facility (SSTF) and a Space Station Control Center (SSCC). The software components of these systems will be developed in a collection of computer systems called the Ground Systems Development Environment (GSDE). The GSDE will make use of tools and procedures developed by the SSFP SSE contractor. Both the SSTF and the SSCC will be developed using both shared and duplicated elements of the GSDE.

The SSTF is being developed under the Training Systems Contract (TSC). The SSCC is being developed under the Mission Systems Contract (MSC). As part of the project planning and development effort, the MSC and TSC contractors have developed plans for CM of software during coding and all stages of testing. At several points these plans have been reviewed by NASA and by this study task, leading to changes and improvements in the CM plans. (Previous assessments by this study are documented in CSC/TM-91/6102, *Interface Requirements Analysis*, and CSC/TM-91/6061, *Operations Scenarios*. See section 1.3 for citations and related documentation).

As the planning continues and becomes more detailed, further reviews (such as this one) will be conducted to ensure that the integrity and reliability of developed software remain high priority considerations. This review is based on a set of goal-centered CM requirements that emphasize the end result of CM without specifying its implementation.

During the Systems Development and Acquisition Phase², SSTF and SSCC software will be configuration-managed using contractor-specified tools in their respective software production environments (SPEs). When the software is transitioned into the Systems Integration and Testing Phase following SubSystem Acceptance Test (SSAT) (or sooner, depending on contractor-dependent integration procedures), it will be placed under formal CM on the Ground Systems/SPF (GS/SPF) using the tools provided by the SSE. Integration testing and build-up to delivery will involve both contractor-managed and the SSE-supplied CM capabilities.

1.1 Assessment summary

During the two years of this study effort, the CM plans of both contractors have changed to reflect changes in the GSDE and have evolved to include more specifics and fewer potential problems. Given the size of the projects, the importance of CM in long-duration system reliability, and the growing emphasis on total quality management, the CM plans of both contractors show serious commitment to professionalism.

¹ Acronyms and abbreviations that are in common use in the Space Station Freedom community are not spelled out in the text, but are defined in the Glossary at the end of this report.

² The MSC/TSC System Development Life Cycle is described in JSC-25519, DA3 Software Development Metrics handbook

Both CM plans, however, fall short of the amount of detail that might be expected at the current stage of these projects. Neither project has chosen to embrace fully the CM capabilities developed by the SSE, nor have they provided equivalent detail about their alternative plans. Since the SSE provides a reasonable baseline for CM (which would allow the projects to establish detailed CM plans with relatively small efforts), the lack of detail is a matter of concern.

The SSCC approach to CM currently involves a non-standard, partly non-COTS set of tools and capabilities (the CORCASE-Atherton proposal) about which little information is available. Although the overall SSCC CM approach is sound, there seems to be considerable risk in the lack of details of the plan. An in-depth, independent assessment is recommended.

The SSTF approach to CM involves distributing CM functionality across three different environments to maximize operational ease of use. The specific CM tools to be used in two of these environments are not yet identified. Since the overall SSTF CM plan emphasizes ease of development rather than product accountability, this lack of specific detail poses a risk in terms of assessing long-term maintainability of the system. We recommend that the operationally convenient distributed CM system should be supplemented with automated procedures that place all delivered software under GS/SPF CM control. Furthermore, when the specifics of the distributed CM system are identified, those systems should be independently reviewed.

1.2 Ground system software development

Ground systems software for the SSCC and SSTF will be developed and tested using combinations of development computers and workstations (collectively referred to as software production environments), a Ground Systems SPF (GS/SPF), and target platforms that are essentially the operational target environments or equivalent. Configuration management will take place in all three environments as software progresses from code and unit test through integration to operational use. Figure 1 shows the basic development context. The target environments include IBM mainframe computers, Unix-based workstations, and mission-specific special-purpose hardware.

Software for the SSTF and SSCC will be developed in the SPEs or in subcontractor facilities, and accepted into the GSDE following SSAT for systems integration and testing. The software is then placed under formal CM on the GSDE host (the GS/SPF). The integration and test will take place in the target environment, or on platforms that are essentially equivalent to the target.

For convenience in this report, the testing environments will be referred to as **Integration and Qualification Test Environments (IQTEs)**. (The two projects use different terms for the target environments. IQTE is a synonym for either term.) Following the Systems Integration and Testing phase, qualification testing (QT) will be performed prior to delivery to operations.

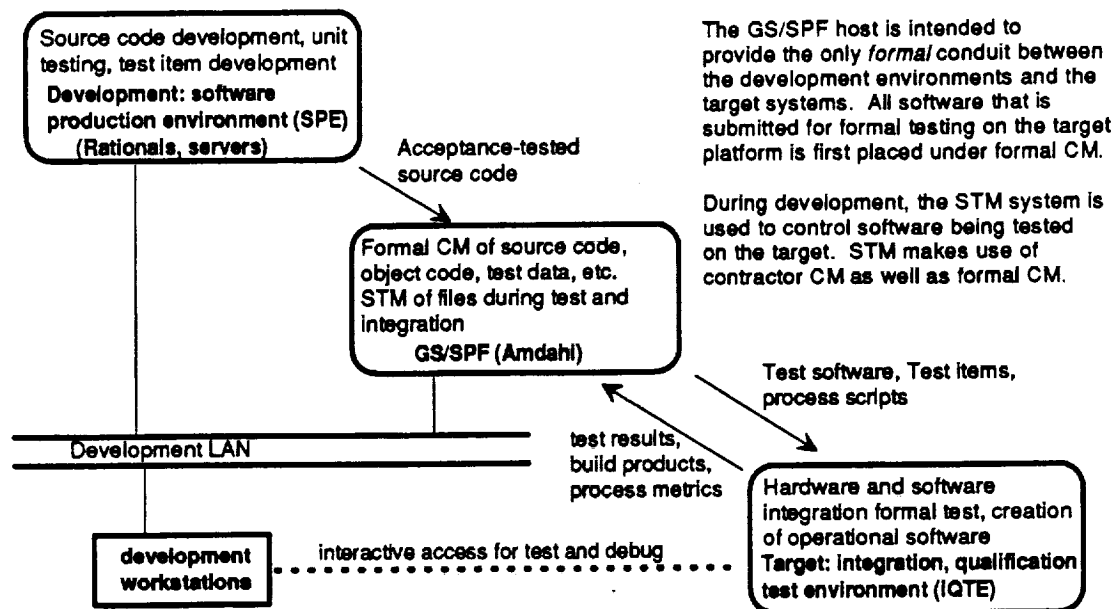


Figure 1. Ground System Development Environment (GSDE)

The GSDE serves as the basic development environment for both the SSCC and the SSTF; only the GS/SPF is shared between the two.

The primary language for development of Space Station Freedom software is Ada*. The SPEs will include Ada-compilation platforms (e.g., Rational R1000 computers) to support Ada development. It is also probable that a substantial amount of non-Ada code, primarily C-language, will be developed (or reused) and supported. Workstations and file servers will be used along with the Ada compilation platforms to support development. Configuration management and software measurement in the SPEs will be performed by the developers, using contractor-specified tools.

The GS/SPF is an IBM-compatible mainframe with a separate instance of the SSE SPF software for each project (SSCC and SSTF). The GS/SPF will host both the formal CM system and the STM system, along with disk storage supporting both systems. Formal CM will be used to manage software following Acceptance Test. For reasons of security and software integrity, the GS/SPF will serve as the conduit for moving software from the SPEs to the targets.

1.3 Organization of Report

Following this Introduction, Section 2 reviews the software CM requirements that form the basis for evaluation. Section 3 describes operational considerations of trying to perform CM in a distributed environment. Section 4 presents a description and assessment of the SSCC CM plans, and Section 5 does the same for the SSTF CM plans. Section 6 presents the recommendations of the study team.

* Ada is a trademark of the U. S. Department of Defense, Ada Joint Program Office

1.4 References and related documentation

Atherton Technology, "Software tools integrate the management of complex design products", from *Computer Design*, August 1, 1991

Atherton Technology, "Software BackPlane", product news release, 1991

Babich, Wayne, *Software Configuration Management: Coordination for Team Productivity*, Addison-Wesley, 1986

CAE-Link Corporation, SSTF Software Configuration Management Approach, September 1991 (briefing)

CAE-Link Corporation, Space Station Training Facility Ground System Development Environment Current Capabilities, December 1991 (briefing)

Card, David, and Glass, Robert, *Measuring Software Design Quality*, Prentice Hall, 1990

Computer Sciences Corporation, *Digital Systems Development Methodology*, May 1990

Computer Sciences Corporation, *Ground Systems Development Environment (GSDE) Interface Requirements Analysis: Operations Scenarios*, CSC/TM-91/6061, February 1991

Computer Sciences Corporation, *Ground Systems Development Environment (GSDE) Interface Requirements Analysis Final Report*, CSC/TM-91/6102, June 1991

Computer Sciences Corporation, *SSE Software Test Management (STM) Capability: Using STM in the Ground Systems Development Environment (GSDE)*, February 1992

Computer Sciences Corporation, *Configuration Management and Software Measurement in the Ground Systems Development Environment (GSDE)*, February 1992

Lockheed Missiles and Space Company, *SSE System Users Guide; Space Station Freedom Software Support Environment (DRLI 23)*, LMSC F255423, March 1991

Loral Space Information Systems, Configuration Management Subsystem: Subsystem Functional Requirements Review (SSFRR) Presentation, October 1991 (briefing)

Loral Space Information Systems, Ground Systems Development Environment: Software Production Environment: Review of Current Capabilities, October 1991 (briefing)

Loral Space Information Systems, GSDE Augmented with CORCASE, January 1992 (briefing)

NASA JSC, *DA3 Software Development Metrics Handbook, Version 1*, JSC-25519, December 1991

Section 2 - Requirements (Analysis Context)

This section summarize the CM requirements published earlier (February 1992). It establishes the context--the accomplishment of CM objectives--for the interface study and the assessments in section 6.

2.1 Goals of CM requirements

Configuration management is a process mechanism* used to ensure that a delivered product is a true reflection of the effort that went into its construction. On a large project, software CM is necessary for the delivery of reliable, maintainable software. CM mechanisms must be applied throughout the development process or they will not be effective.

The assessment of CM, however, can focus either on the continuing mechanisms (a *process view*) or on the end-result of those mechanisms. This study is based on the latter view.

The SSE CM requirements and design documents provide an example of the process-oriented view, with detailed specifications of processes, formats, and relationships. Typical users guides and standards and practices manuals also embody this view. Because of the diversity of the systems being developed, and the range of options available to achieve effective CM (see section 2.4), this view seemed to be too constraining for the RICIS evaluation. Accordingly, the end-item view of CM was developed to support assessment of different approaches based on expected results.

The end-result view of CM focuses what CM is intended to accomplish, rather than on the mechanisms involved. This goal-oriented view prescribes the conditions of controlled, identified, traceable software products without specifying the details of the process.

The requirements developed in the earlier study report (CM and SW Measurements in the GSDE) and summarized below are based on this end-item, goal-oriented view of software.

As a process, CM is valuable (essential, in our view) for efficient management of software development. That is to say, CM would be cost-effective for large projects even discounting its value in assuring product quality. Both the MSC and the TSC contractors have described their plans to use CM in this manner. That use of CM, however, is a contractor prerogative and is not addressed in this assessment.

* The term "process mechanism" is used in contrast to "product mechanisms" which are applied to a product at the end of a process rather than during its course.

As a product quality assurance mechanism, a CM system must be able to demonstrate that the delivered product includes exactly those versions of components that were tested and approved. It must also be able to demonstrate that all test certifications claimed for the product were actually performed on the software being delivered (and not on some undelivered version of the product).

In order to support sustaining engineering of the product, the CM system must ensure that the product can be regenerated from files that are write-locked (that is, files that can be read but not changed). It must also support traceability to the set of requirements, interface specifications, and design documents that were in effect at the time of delivery.

This report is based on assessments of the SSTF and SSCC plans for CM in the context of those desired end-item conditions. The requirements summarized below formed the basis of the assessment (with some reference to the earlier, higher-level set of requirements described in the June, 1991 Interface Requirements Analysis). The assessment (and the requirements) focus on the goals to be achieved, rather than on the mechanisms chosen to achieve them.

2.2 CM Requirements Summary

The numbered items in sans-serif text are the requirements. The indented paragraphs under each requirement are explanatory material and not part of the requirements specification.

1. Source files for all delivered software must be placed under NASA-managed CM.

When you deliver software to operations: you must place the source code and any ancillary files for all developed software under NASA-managed CM (e.g., the SSE-provided CM system on the GS/SPF). This does not include mission specific data, which is under separate control, or unmodified COTS software. You must also record the steps (e.g., compilation, linking) that were followed (or can be followed) in building the deliverable software from the controlled files.

2. The software that is delivered must be the same as the software that was tested.

The QT process certifies software as having passed certain tests and therefore as meeting the associated requirements. You must keep records to show that the software used in the QT process was built from the same source files that are used for the delivery. You must be able to show that the same build options (e.g., optimization settings used during compilation) were used both for QT and for delivery.

3. All files necessary to recreate the qualification tests for delivered software must be placed under NASA-managed CM.

All test scripts and test data used in QT must be write-locked and stored, available to support retesting or regression testing. The STM mechanism (provided by the SSE on the GS/SPF) provides snapshot and archive mechanisms that would be suitable for meeting this requirement.

4. Configuration control mechanisms for software in the Systems Integration and Testing phase must involve permanent records and independent authorization.

Once software is transitioned to SIT and QT, all changes must be recorded. This requirement says that the person approving a change cannot be the same as the person initiating and performing the change, and the approval must be recorded. The SSE CM system provides appropriate mechanisms for this approval and recording. Non-SSE CM must provide some equivalent mechanisms.

5. The status of all approved changes shall be recorded and accessible.

For any approved change, it must be possible to determine which components will be affected, and which versions of those components. The status (e.g., applied, deferred, not applicable) of a change to a particular source code file must be available (that is, accessible to inquiry or included in a report). The change status of components in snapshots of systems should also be accessible.

6. The status of all DRs and STRs shall be recorded and accessible.

From a sustaining engineering perspective, the relative stability of components is a valuable indicator of future reliability. The status of discrepancies (degree of criticality, components implicated, resolution, identification of test case or activity that triggered the report) provides an avenue to assessment of system quality. As long as *all* discrepancies are recorded, the *absence* of discrepancy reports can be taken as a positive indicator of the quality of the system.

7. Applicable interface specifications shall be archived with software delivery snapshots.

Interface specifications, like requirements, are subject to change. Even without requirements changes, there are shifts of functionality between hardware and software, between developed software and COTS, and between developed subsystems and systems. In order to perform sustaining engineering on a fielded system, it is important to have

recorded the interface agreements that existed for that system at the time of delivery.

2.3 Space Station SSE CM support

The GS/SPF provides the Oracle-based SSE CM system for formal control of software. This CM system is described in the *SSE System User's Guide*, DRLI #23. The CM system uses the GS/SPF file mechanisms for storage of data, and the Oracle database management system to maintain information about configuration items. The CM system provides mechanisms for:

- configuration identification,
- change-instrument processing,
- component checkin and checkout,
- snapshot and archive management,
- access control, and
- status reporting.

Most functions of the CM system are available in both interactive menu and command-line batch modes of operation. The CM system is intended to interoperate with developmental CM systems (such as the rational CMVC system), and provides mechanisms for transfer and operations on collections of components as well as on individual components.

The SF/SPF will also provide an Oracle-based Software Test Management (STM) capability to manage files and test scripts during integration and qualification testing. STM provides a mechanism for moving software between the development and the target environments in a controlled fashion. Through the use of command-line operations, STM can interoperate with the SSE CM system and with any developmental CM that supports command-line (non-interactive) processing.

STM provides a controlled environment for defining tests, assembling testbeds and test articles, and recording results of testing. (The testing itself is on the target platform, and is not under control of STM.) STM also provides a mechanism to record snapshots and build information. It is anticipated that the STM build mechanisms will be translated into the formal build-delivery process to be provided in OI 7.0 of the SSE.

Together with contractor-developed procedures and automated mechanisms, the SSE CM and STM capabilities provide tools that can be used to satisfy the requirements summarized above.

2.4 Options and alternatives

There is no single way to achieve the goals identified for CM in the GSDE. Given the range of different computers, workstations, servers, application characteristics, and languages of available reuse libraries, multiple approaches are called for. The need to integrate mainframe-based Ada code with workstation-based C and Ada code requires either multi-platform CM support or multiple CM systems. The tools provided by the SSE (STM and the CM system) provide most of the required capability, but not all of it.

During the SI&T phase, software will be uploaded from controlled storage for compilation on the target platforms. The compilation products may be kept in the target environment for extended periods, to minimize file transfer and recompilation time. These products need to be controlled in the target, as they will be used for formal testing.

At the same time, software may be uploaded from the developers' workstations to the target for developmental (e.g., unit) testing on the target. The target-based CM system needs to be able to distinguish between developmental and controlled software, lest deliveries inadvertently include untested code. The SSE-provided tools provide only limited capability to deal with this situation.

Variations of procedural controls and automated CM tools will be needed to effect control in this diverse development-CM-target environment. This diversity is the reason for using goal-oriented rather than process-oriented requirements in assessing the MSC and TSC plans for software CM.

		NO. 11		NO. 12		NO. 13		NO. 14		NO. 15		NO. 16		NO. 17		NO. 18		NO. 19		NO. 20		NO. 21		NO. 22		NO. 23		NO. 24		NO. 25		NO. 26		NO. 27		NO. 28		NO. 29		NO. 30		NO. 31		NO. 32		NO. 33		NO. 34		NO. 35		NO. 36		NO. 37		NO. 38		NO. 39		NO. 40		NO. 41		NO. 42		NO. 43		NO. 44		NO. 45		NO. 46		NO. 47		NO. 48		NO. 49		NO. 50		NO. 51		NO. 52		NO. 53		NO. 54		NO. 55		NO. 56		NO. 57		NO. 58		NO. 59		NO. 60		NO. 61		NO. 62		NO. 63		NO. 64		NO. 65		NO. 66		NO. 67		NO. 68		NO. 69		NO. 70		NO. 71		NO. 72		NO. 73		NO. 74		NO. 75		NO. 76		NO. 77		NO. 78		NO. 79		NO. 80		NO. 81		NO. 82		NO. 83		NO. 84		NO. 85		NO. 86		NO. 87		NO. 88		NO. 89		NO. 90		NO. 91		NO. 92		NO. 93		NO. 94		NO. 95		NO. 96		NO. 97		NO. 98		NO. 99		NO. 100	
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100																																																																																		

Section 3 - Operational Considerations

There is a constant tension in any large software development effort between the *flexibility* needed for responsive implementation and the *control* needed to ensure project completion. This tension is evidenced in the operational tradeoffs that are made between complete freedom for developers and rigorous control for the purpose of quality assurance. Programmers argue that rigorous control is antithetical to creative programming, while project-level managers argue that maintainability is equally as important as creative solutions.

Operationally this tension leads to a range of CM solutions. The formality of CM varies from minimal control on the developers' own workspaces, through moderate control when a subsystem enters testing, to rigorous control when a product undergoes QT or delivery. This range of CM solutions, incorporated in both MSC and TSC plans, provides a mechanism for escalating the level of control throughout development. However, it also provides a mechanism for subverting the intent of CM by applying too lax a standard for transitioning to more formal control.

There is a risk in any schedule-dominated organization, like the manned space program, of magnifying the importance of local milestones. When programmers face deadlines with almost-working code, they have a tendency to postpone or ignore requirements for CM in favor of getting the software to work. The long-range implications of poorly-controlled software do not seem nearly as immediate as the schedule that calls for a delivery *now*. It's the job of the QA organization and established procedures to provide a countervailing influence.

Both the SSTF and the SSCC projects include mechanisms for graduated CM during the life cycle. In this section we take a look at some of the operational considerations involved in implementing those mechanisms with either automated or manual procedures.

Automated procedures are those which basically don't require the programmer (or manager) to *choose* whether or not to utilize available CM mechanisms. *Manual procedures*, in contrast, rely on personal discipline; they permit the developer to bypass controls. As an example, the SSE-provided CM system *requires* a change instrument for any change to controlled software. The SSE-provided STM capability, on the other hand, makes use of externally controlled software and inherits whatever level of control (or lack thereof) is provided by the owner of the software. In this regard formal CM is automated while STM uses manual procedures.

In this section (and in this study) we are concerned with configuration management at the interface between the development environments and the testing environments. Originally that interface was completely defined within the GS/SPF, but as the two projects have evolved alternative interfaces have been identified. The following subsections address the typical development process, the basic mechanisms of control, and the possibilities and implications of bypassing such control.

Source files are translated (if necessary) and stored in libraries of software objects, including test data objects. These objects are used to create executable files that are combined into a specific testbed for testing. (The graphic shapes in figure 2 were designed to indicate the "fitting together" of test item, test data, and supporting testbed.)

As indicated by the feedback arrows going right-to-left, there are specific products generated in testing that can be retained for subsequent use. These products include executables from the testbed as well as test data that is output from one test and used as input to another. There is also a feedback of information that affects all of the source-file column of software components.

The same process, using mission data instead of test data and deliverable software instead of scaffolding, will be used to generate operational software loads.

Figure 2 portrays the basic steps that are of concern in configuration management of the development-to-test interface. These elements (source files, libraries, products) are further discussed in the remainder of this section.

Figure 3 shows the symbols that use used in figure 2, along with other that will be used in later figures to describe possible CM operations. Basically, heavier borders are used to indicate greater formality of configuration control, and darker shading is used to signify later steps in the integration and test process.


Configuration control formalism	Genesis of files
These symbols are used to indicate the potential for unrecorded or unapproved change to a component (a file).	These symbols differentiate primary files from derived files (e.g., compilation products), Files (e.g., COTS S/W) from other organizations are separately classified.
<div data-bbox="298 1318 493 1402">controlled by developer</div> <div data-bbox="516 1318 776 1444">files are managed by the developer without separate approval or record-keeping</div>	<div data-bbox="867 1318 1062 1402">source</div> <div data-bbox="1084 1318 1338 1402">source code, scripts, test data, etc.</div>
<div data-bbox="298 1457 493 1549">moderate control</div> <div data-bbox="516 1457 776 1549">independent CM but limited review and physical file security</div>	<div data-bbox="867 1457 1062 1541">intermediate product</div> <div data-bbox="1084 1457 1386 1583">generated from source by compilers or other processors; object code, libraries, test data, etc.</div>
<div data-bbox="298 1596 493 1688">formal control</div> <div data-bbox="516 1596 776 1696">rigorous CM; file changes performed only by CMO</div>	<div data-bbox="867 1596 1062 1688">deliverable product</div> <div data-bbox="1084 1596 1386 1696">executable images, generated tables and data (e.g., screens), etc.</div>
<div data-bbox="315 1738 477 1814">library</div> <div data-bbox="516 1738 753 1856">controlled files are provided by other organizations (e.g., COTS, flight S/W)</div>	<div data-bbox="857 1709 1094 1814">  </div> <div data-bbox="1110 1709 1370 1856">testbed configuration generated from test article, test data, and supporting software</div>

Figure 3. CM process symbols.

Heavier borders are used to indicate more formal configuration control. Darker shading indicates products closer to the delivery stage of software.

3.2 Clean-slate approach to formal CM

At the start of a new phase of a development effort, all the myriad elements are coordinated. There is consistency among source files, libraries, test data, command files, and testing plans. It is a simple matter to determine what software has been tested, and what data files were used.

As development and testing progresses, a sort of entropy sets in. Software is iteratively tested and modified by different developers on different schedules. Test data files evolve as special cases are added to test additional requirements. Programmers share command files and test scaffolding, both to save time and to make use of tested workarounds. Precise knowledge of what software is being tested with what version of test data gets harder and harder to come by.

To address this problem, development leaders often resort to periodic zero-basing of the project or subsystem. All source files are frozen and recompiled to generate defined libraries and executables. Redundant copies and minor variants of command files are deleted from common access. Everyone starts fresh from a common baseline.

Experience with mainframe-centered NASA ground systems development has shown the effectiveness of this approach even with FORTRAN-based projects. Source code may be developed on the mainframe or on workstations; when ready for integration testing it is added to a source-control library that supports maintenance of controlled and "test" versions of software. The source is compiled on the mainframe into load modules which serve as testbeds for further testing.

These load modules can be used as libraries of object code by using linkage editing tools. (This capability is similar to the functioning of Ada libraries that include source and intermediate code for use in constructing executable images.) Test articles are link-edited into controlled load modules for testing, avoiding the need for massive recompilation or build procedures. When combined with a source library control system that supports both development and test versions of code and data, this approach has been proven on a large number of ground system software efforts.

Even with source code and access control, however, the status of libraries and load modules gradually becomes uncertain due to the non-automated procedures involved in making changes. Since every programmer needs access to the "controlled" files, questions arise as to which versions of fixes and workarounds have been applied to the official version of the load modules. At that point, as often as weekly during heavy testing and debug efforts, the old load module is erased and a new load module is generated from controlled source. All necessary fixes are first applied to the source code to ensure that the new baseline is up-to-date.

Developers thereby start from a new "zero-based" baseline.

On activities as large as the Space Station Freedom ground support systems, it will not generally be practical to zero-base an entire project or subsystem. Both the SSCC and the SSTF involve distributed systems with multiple parallel development efforts, complex real-time interactions, and integrated project schedules. Unlike the mainframe-centered experiences described above, there may not be a single point or a small set of items to erase and recompile. Mechanisms and approaches are needed to meet the requirements summarized in section 2 for reliability and maintainability of ground system software.

One of the primary mechanisms for quality assurance of the software is the use of separate computer systems for development, and for integration and test. The transition from SPE to IQTE can serve the same function as the "zero-basing" described above. This section discusses ways that the independence of the IQTE can be used to good advantage.

Figure 4 describes the basic "clean-slate" approach to using the IQTE as a consistency and reliability mechanism.

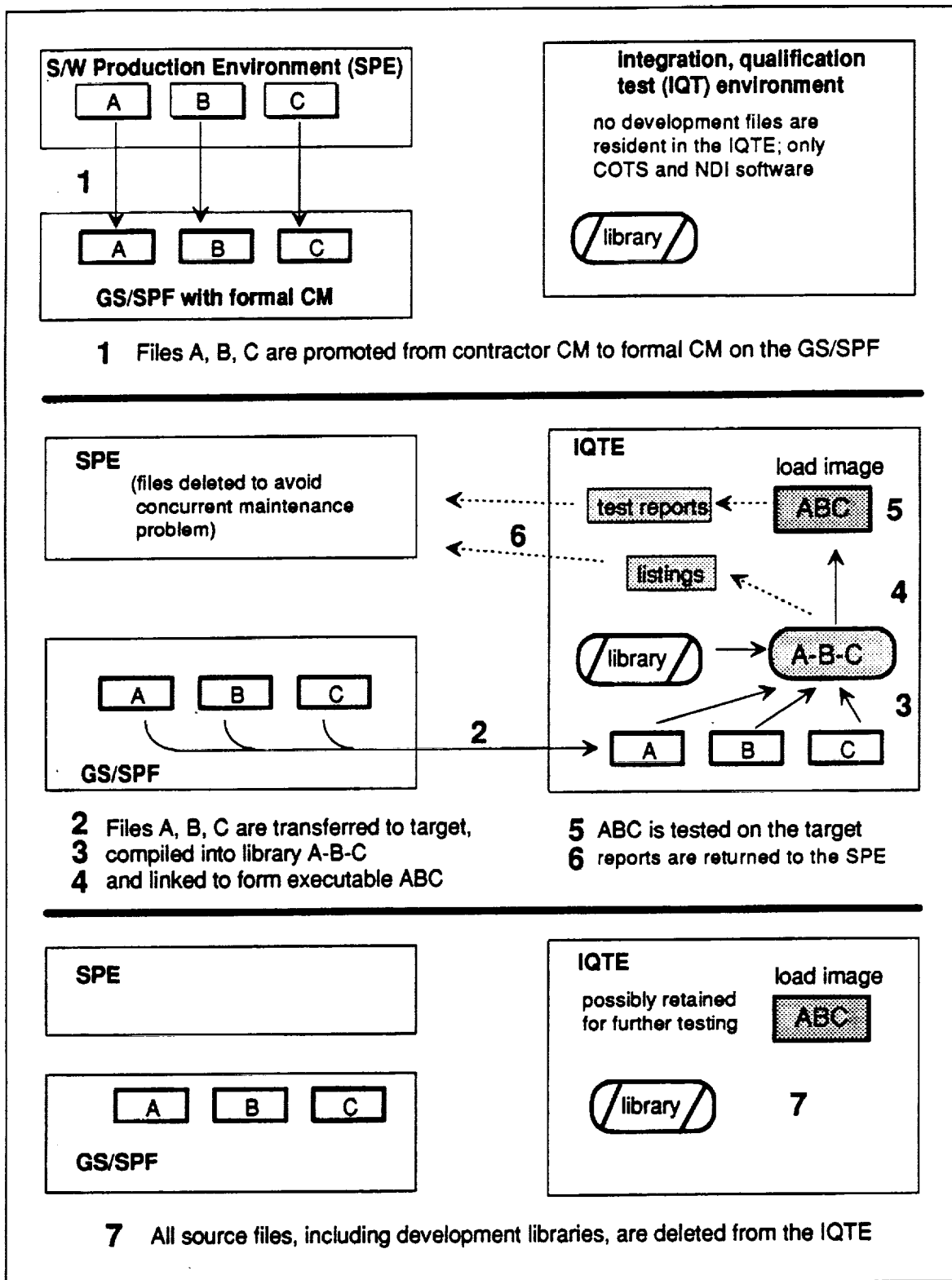


Figure 4. Clean-slate approach to formal CM

Very high reliability is achieved with some performance penalty by erasing all product files after use. The CMO places source code under formal control, uploads it to the IQTE for test, then deletes all products after testing is complete.

The starting point for this approach is the set of source files under contractor-managed CM in the SPE. The files that are ready for testing (whether IT or QT) are checked in to formal CM, using the SSE-provided CM procedures. Each component to be checked in must be identified on a change instrument. (A single instrument can be used to check in an entire subsystem, if desired). The files are copied to the GS/SPF by the developers, and turned over to the CMO who performs the actual checkin. Note that the only ground system files on the IQTE are "non-developed items" such as standard libraries (e.g., COTS software) and software from other Space Station activities.

Once the files are checked in to formal CM, they should be deleted from the development library to avoid any chance of inconsistency between duplicate copies.

Using scripts provided by the developers, the CMO copies specific files to the IQTE. Still under script-based commands, the files are compiled into a library for use in generating executable files. (There may be more than one library involved, and more than one executable image.) The compile and build processes produce listings and other information output which are returned to the developers.

The development products are tested on the IQTE, generating test reports for use in certifying the software components tested.

At the completion of testing, the product files are deleted from the IQTE. The primary consideration is that all source files and compilation libraries be erased, so that the next test sequence does not inadvertently reuse the products of this test. If specific executable images are required to support testing of other products, they may be retained (locked, of course, against any modification) on the IQTE.

A primary characteristic of this approach is that *software* flows from the controlled environment *to* the IQTE; only information is fed back. This ensures that files on the GS/SPF are always secure. Both MSC and TSC have described their CM plans along these lines.

This "clean slate" approach ensures that all test items are generated from controlled source code, but exacts a penalty in performance. Copying all of the necessary files can become a significant overhead, and massive recompilation is even more costly. For large subsystems (e.g., 50-75 K lines of code), recompilation time could become a significant schedule bottleneck.

It should be noted that a typical load image generated for integration or qualification testing might remain "under test" for an extended period of time (e.g., weeks). The need for recopying and recompilation is not an everyday requirement. Since copying and compilation can often be performed during off-peak hours, the performance penalty for this approach is not so great as it might at first appear.

This approach has the advantage of ensuring that "inadvertent reuse" does not occur, but, like other aspects of the development process, is subject to cost-benefit analysis.

3.2.1 IQTE-resident product files

A modification of the clean slate approach that addresses some of the performance issues is illustrated in figure 5. While it is still the case that no software flows back to the GS/SPF, some library files are retained in the IQTE to alleviate the recompilation problem.

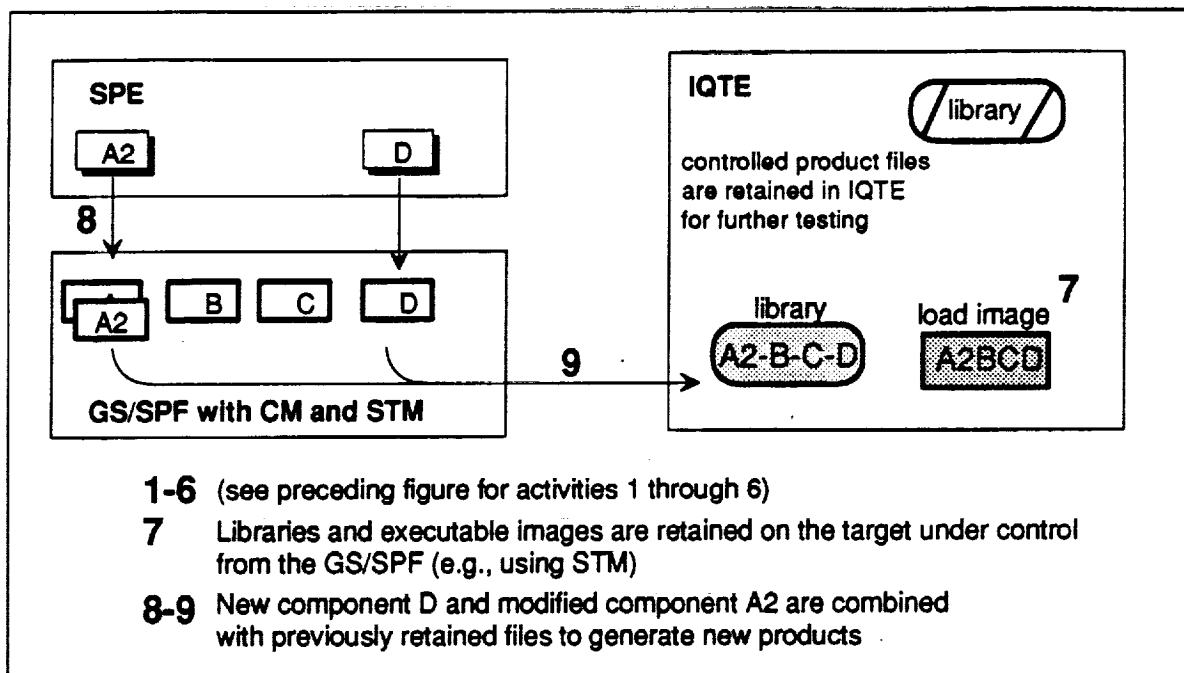


Figure 5. IQTE-resident product files

Improved performance is achieved with only moderate compromise of security by retaining compilation products and associated source files in the IQTE.

Because of the way advanced compiler systems operate, it is impractical to keep only object files in a compilation library. Ada compilation systems in particular require that source code be accessible in order to verify interface definitions. While C compilers do not require source code for related files, typical SCCS and Make systems do. Accordingly, the libraries noted in figure 5 represent both source (write-locked, preferably) and object code.

The intent of retaining these libraries on the IQTE is *not* to support editing and debugging. Rather it is for the purpose of uploading and compiling other files which may require the earlier libraries in order to compile. By retaining write-locked source files and object code on the target, it is possible to avoid having to recopy and recompile everything for the sake of a small number of new source code components. Editing of source files on the target machines should be strictly prohibited, even for unit-test files. (This is the reason for write-locking the files.)

The MSC CM plan states that this will be the case. The TSC plan is less definite, and permits debugging of files that are being unit-tested in the IQTE. The use of a Rational

R1000 as part of the IQTE would facilitate editing in the target environment, even if such modifications are officially discouraged.

As a cautionary note: developers on the AAS project for the FAA* found that, when subsystems were developed on Rational computers and uploaded to IBM mainframes for compilation and test, massive recompilation was sometimes *preferable* to piecemeal changes and updates. The cost of determining which routines needed recompilation, using the IBM SLCM configuration management tool, was a significant factor in the tradeoff analysis.

3.2.2 GS/SPF-resident product files

Leaving product files (especially libraries containing source code) in the IQTE poses risks of inadvertent use. Another option that reduces the cost of recompilation while retaining control of product files is to transfer those files back to the GS/SPF under configuration control. Figure 6 illustrates this procedure.

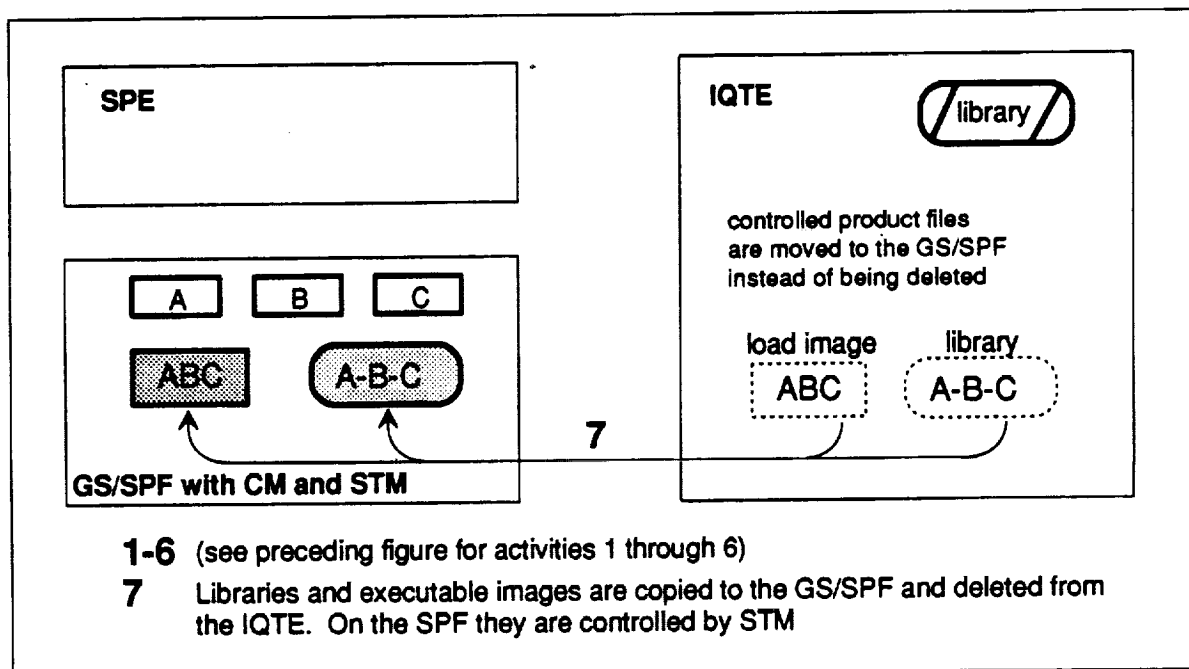


Figure 6. Storing product files in the GS/SPF

Performance and security are balanced by keeping compilation products under formal CM, available for use in subsequent test activities.

The GS/SPF provides the most rigorous configuration control in the entire GSDE. The procedure suggested here would place product files (executable images, test data, compilation libraries, etc.) under configuration control in snapshot form--able to be used, but not modified. The primary disadvantage to this procedure is that a conduit is opened

* The Advanced Automation System is an Ada-language real-time command and control system being built on IBM mainframes using Rational development tools.

for files to be copied *from the target to the development environment*. This risk, however, is quite manageable using the snapshot mechanism. The SSE-provided STM has mechanisms in place to receive and control such product files without making them accessible to editing.

At present, neither TSC nor MSC has indicated any intent to use this procedure. The MSC approach does not permit any reverse flow of files from the target to the development host. The TSC approach involves maintaining these files in the "reconfiguration host" in the target environment.

3.3 Combined formal/informal CM

With STM, the GS/SPF has the capability of supporting a less formal method of configuration control, as a complement to the formal CM procedure. This Software Test Management capability provides for close integration between contractor-managed CM and the GS/SPF. Figure 7 illustrates the use of this capability to combine formal and informal source code control.

It will often be the case that developers use tested, certified software as scaffolding for newly developed code. It is often easier and more reliable to use the actual target of an interface rather than a test version composed of stubs. Unit testing, therefore, can combine certified code with newly developed code. STM provides a mechanism for placing unit-test software under sufficient control to allow it to coexist safely with formally-controlled code.

This procedure requires that compilation products from other activities be retained under CM either in the GS/SPF or in the IQTE. The existing products (e.g., libraries and executable images) are combined with STM-managed source files to create unit-test configurations.

The primary risk inherent in this approach is that product files (such as the compilation library *A-B-C-d-e* in figure 7) that include combinations of accepted and unaccepted code may inadvertently be used for formal testing. In figure 7, for example, if the library *A-B-C-d-e* were used instead of the original library *A-B-C*, the presence of not-yet-accepted components *d* and *e* might mask errors that would otherwise be uncovered during testing.

The STM solution to this risk is to ensure that all product files are properly identified as resulting from specified tests. In this example, the library *A-B-C-d-e* could not be checked back in to replace *A-B-C*, because product files cannot be updated or replaced except as part of a specific test procedure.

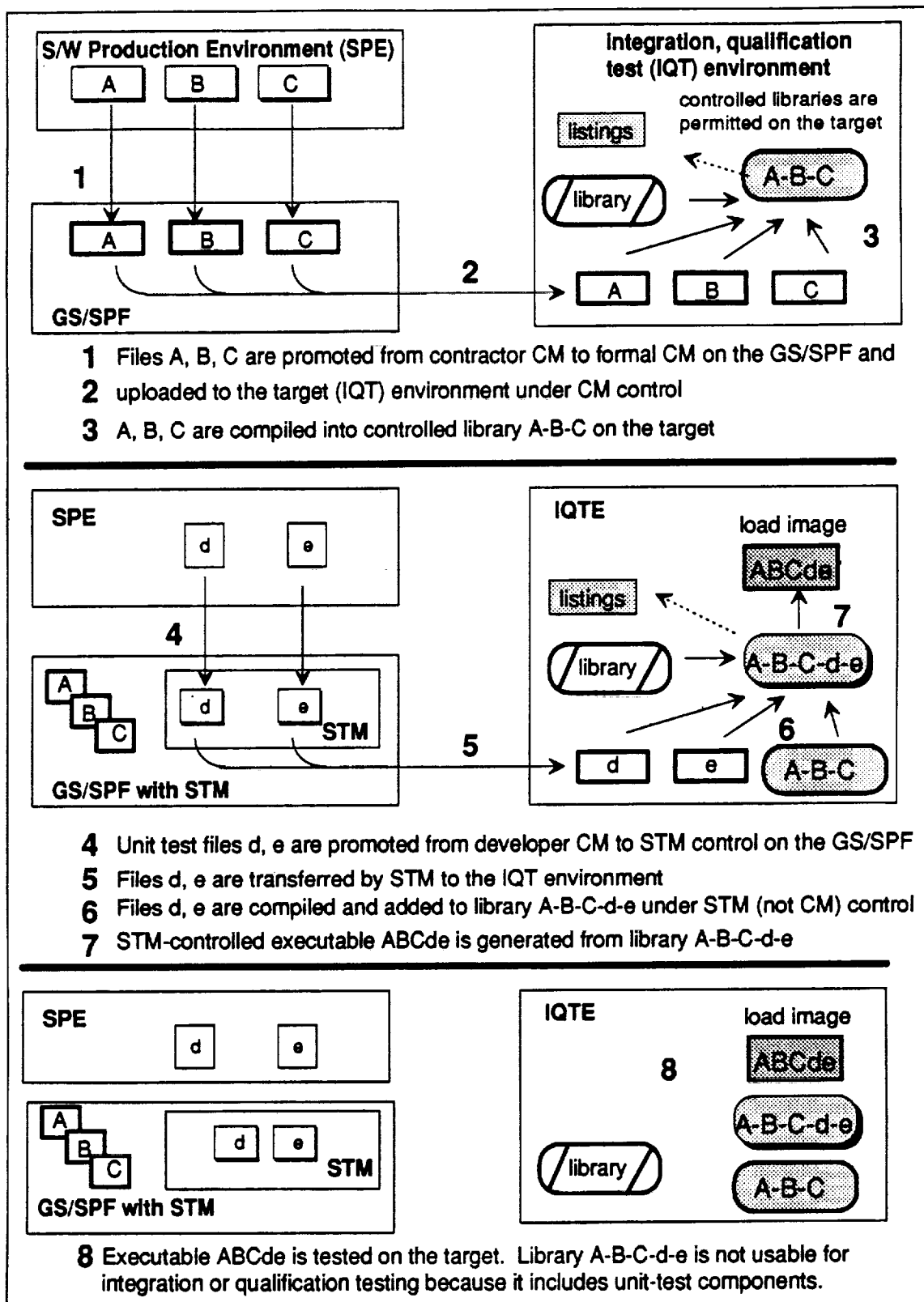


Figure 7. Combining formal and informal CM

File access and system performance is aided with the provision of informal mechanisms for software testing. File integrity is maintained using the SSE-provided STM.

3.4 Using the SPE as a conduit

Thus far the discussion has assumed that the GS/SPF is the sole conduit between the SPE and the IQTE for post-acceptance software. (Software undergoing unit test is not addressed by this study). However, it is also possible for software to be transferred directly from the SPE to the IQTE, as shown in figure 8.

Source files would be promoted from contractor-managed CM to formal CM on the GS/SPF, as was the case in figure 4. However, the files would be retained in the SPE as well, in order to be uploaded to the IQTE directly from the SPE. Compilation on the target would not be directly based on the files under formal control.

As a result of build and test activities on the target platform, there would be several copies of basic or derived files under various levels of control. The *same* files would exist on the SPE in contractor-managed CM, on the GS/SPF in formal CM, in the IQTE as source files, and perhaps in compilation libraries as well. (It's also likely that the programmer has retained a copy of his or her files in private workspace, but that should not pose a problem for CM). The real risk, of course, is that all of these files would *not* be the same even though the CM system would record it so.

As suggested in figure 8, it would be entirely possible for changes to be made in the SPE (under contractor CM) and propagated to the IQTE without updating the formally controlled files. The result is a compilation library and executable image (e.g., A2BC) that do not correspond to the controlled source code. If those product files are delivered, or used in other testing, the overall reliability of the development process is significantly affected.

The CM plans of MSC and TSC both include some mechanisms for bypassing the GS/SPF in promoting files from the SPE to the target. The MSC proposal using CORCASE would provide a distinct SPE-to-target interface mechanism. The TSC proposal would effectively treat the reconfiguration computer as an extension of the SPE when convenient, and an extension of the target at other times.

There are at least two possible approaches to mitigate the risks inherent in this interface mechanism. One is to require that all compilation and build for test and delivery be performed from GS/SPF storage. This would ensure that even if there were CM failures leading up to testing, the final test or delivery would meet the requirements in section 2.

Another approach is to provide an automated interface between contractor CM and formal CM to report whenever a source file is uploaded to the target from the SPE. This interface would compare the version identification of the uploaded file with the appropriate formal CM record, and if necessary record that the formally-controlled file was no longer valid. It is recommended that this be an automated interface.

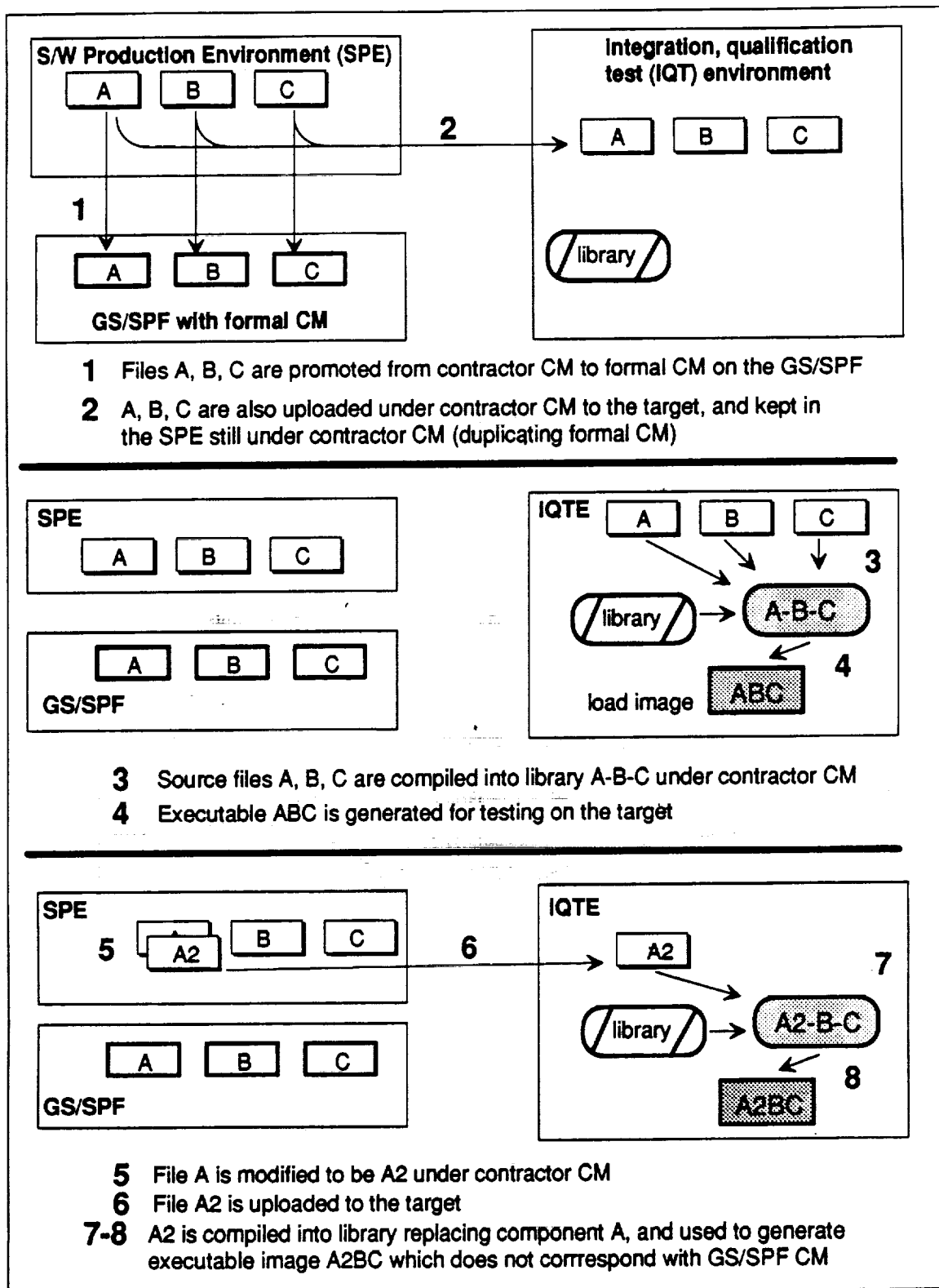


Figure 8. Using the SPE as a conduit

Files on the GS/SPF become redundant and difficult to certify as original and modified source files are moved directly from development to target.

3.5 The mosaic effect

As noted in the preceding discussions there will be a range of formality in the CM imposed on files in the IQTE. Unit testing, acceptance testing, IT, QT, and buildup for delivery will all occur in the same environment. This raises concerns for two types of inadvertent mixture of components.

The term *mosaic*, borrowed from genetics, refers to a composite that is made up of fragments of different origin. In this situation, it describes a test item or configuration whose origin involves different versions of software. ("Mosaic" also sounds better than its less precise alternative, "mongrel".)

3.5.1 Compilation libraries

During development, any given subsystem will require and utilize elements from other subsystems in developing and testing interfaces. This is particularly true in the case of Ada, where interface definitions are owned in common between subsystems. The situation illustrated in figure 9 can easily occur.

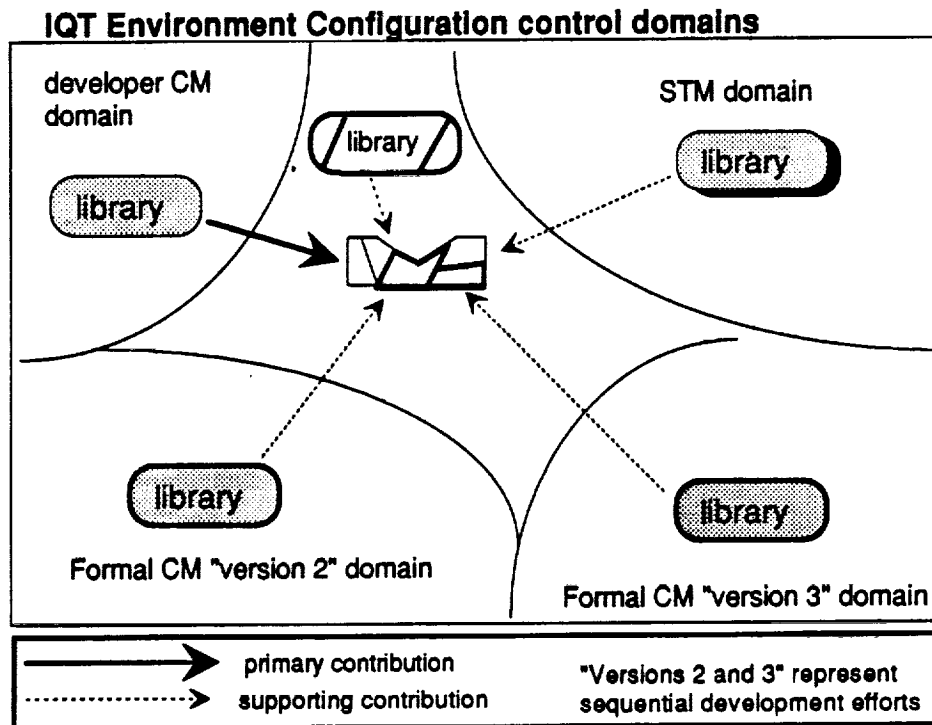


Figure 9. Mixed libraries in the IVT

Since developers will often need access to latest versions of code, sharing of files is inevitable and difficult to manage.

During early testing of software, developers will routinely borrow code (e.g., Ada package specs) from other libraries to ensure consistency and save effort. Similarly, to save effort, these borrowings will become embedded in the command scripts that create

test articles. The availability of multiple libraries serves a valid purpose in facilitating inter-subsystem and inter-version consistency. It also raises a risk that what actually gets *tested* is not exactly what was *planned*.

One solution is to prohibit the implicit sharing of files, and to make explicit sharing subject to formal record-keeping. The goal is not to prevent sharing, only to manage it.

An additional protection is to use file and library access controls to ensure that test software is only generated from its own approved libraries. When a component is transitioned from one level of testing to another, it will have access to a different set of libraries. (This may result in some consternation when a module that worked in unit test won't even compile during integration testing, but that's one of the reasons for integration testing.)

The MSC plan for CM has not yet specified policies for the use of libraries in the CORCASE CM environment. The TSC plan envisions the use of procedural and access-control mechanisms to address the problem.

3.5.2 Testbed construction

A more complex problem occurs in the construction of testbed configurations for testing in the IQTE. The library-sharing described above is largely a matter of convenience during development; it can be managed with access controls on files. During integration testing, however, the combination of resources from different development streams is inevitable. Since different developments occur on different schedules, the situation shown in figure 10 is likely to be the rule rather than the exception.

The primary concern with shared testbed elements, of course, is the impact on repeatability of the testing process. The various development streams constitute moving targets, and reconstruction of all of the elements can be impossible without careful management.

Test data is particularly subject to continuous evolution, both to accommodate more requirements and to reflect changes in the mission statement (e.g., different launch profiles). But *all* of the testbed items are mutable, even when the various subsystems become operational. The recommended approach to managing the mosaic testbed problem is a combination of snapshots and detailed record-keeping.

Snapshots of test configurations improve the probability that regression testing can be performed at a later date. (They don't provide an absolute guarantee, if for no other reason than the possibility of changes in the hardware configuration). Snapshots are particularly useful in the process of tracking down the point when a fault was introduced into a system.

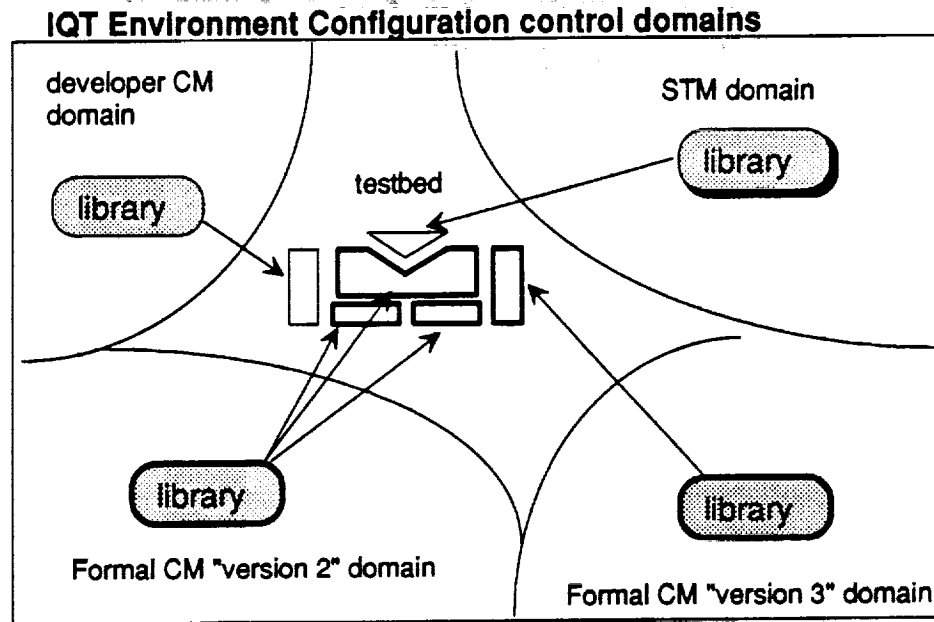


Figure 10. Mixed product files in the IQTE

With several types of testing (unit, acceptance, integration, qualification) occurring in the same environment, sharing of files is inevitable.

Careful record-keeping simply involves tracking the provenance of all items that enter into a testbed. For software, these records will generally provide access to the source code and permit regeneration of the items. For test data, detailed records will at least indicate when a particular data set was created and perhaps how it was generated.

The SSE-proposed build-up process, planned for OI 7.0, will involve detailed information on all stages of the compile and build sequence. That same information, captured for test articles and testbed elements, can provide the information needed to meet the requirements in section 2 concerning reproducibility of testing.

Neither MSC nor TSC has provided sufficiently detailed CM plans to determine how--or whether--this problem will be addressed.

Section 4 - Space Station Control Center

The Space Station Control Center will be a complex of IBM mainframe computers, Unix-based workstations, and special-purpose communications hardware networked to each other and to NASA communications systems. It is being developed for JSC on the Mission Systems Contract. The overall software architecture of the SSCC was described in earlier reports in this study: Appendices B and E of the Interface Requirements Analysis Report (June 1991) provide a good overview.

The SSCC will be developed with a substantial amount of non-Ada code in order to capitalize on previous control center development work. Most of the non-Ada code will be C-language code, and will be used in the consoles (workstations) of the SSCC. Development approaches have been defined to accommodate this dual-language approach.

Overall, the approach of the MSC contractor to software CM has been to follow the requirements and recommendations made by JSC, including the SSE CM capabilities and the recommendations of the RICIS study.

4.1 Overall software workflow

In January, 1992, the MSC contractor proposed an alternative structure for the SSCC Software Production Environment. This alternative replaces the Rational R1000-based development environment with a VAX-based system, and replaced SSE-developed or provided software with custom or different off-the-shelf software. Not all COTS software is changed in the MSC proposal, but the basic development structure is significantly changed.

The primary argument made for this alternative is the need to integrate existing (C-language) software into the SSCC development. The SSE-defined system is almost totally Ada-directed, and was considered unsuitable for the SSCC project. (Note that the flight-software orientation of the SSE is also a consideration, and has been in a factor in this RICIS study effort.)

The alternative was described in the January briefing, "GSDE augmented with CORCASE". A proprietary software management system called CORCASE, developed by Loral's Western Development Lab, would function in the software repository role held by the Rational R1000. The Ada compiler from Verdix would replace the Rational Ada compiler, and the Software Backplane from Atherton Technologies would provide the tool integration capabilities currently provided by Rational and SSE-developed software. Many of the specific tools, such as Teamwork and Interleaf Publisher would be retained in the alternative.

CORCASE, the software repository, was developed for Unix systems to support the DoD software development standard 2167A. It would be modified to run on VAX Ultrix servers and to match JSC-specified life-cycle definitions.

4.2 Configuration management

The CORCASE system would provide software CM services extending from developers' workstations to the test and integration environment. It would interface with the formal CM residing on the GS/SPF. The Rational CM system (CMVC), with its detailed Ada code integration capabilities and subsystem management ("frozen views") approach, would be partially replaced by the combination of CORCASE and the Atherton Software Backplane.

The proposed solution would provide much better support for non-Ada code, because all source code would be managed in the same form. (The Rational R1000 is not a very congenial environment for non-Ada code). Some of the benefits of the R1000 would, of course, be lost, most notably the incremental compilation capability.

The SSE project has developed a substantial amount of software to integrate the Oracle-based CM with the R1000-based CMVC. This software goes considerably beyond the capabilities provided by Rational. This software automates significant portions of the checkin-checkout and verification process for initial entry into the CM system and subsequent entry of changes.

The MSC contractor has proposed to construct a batch file interface using the same capabilities on the GS/SPF and newly developed interface support in CORCASE. The difficulty involved in this construction would presumably be on the same order of magnitude as the original development by the SSE project.

Although details are not clear, it appears that the CORCASE system would support direct, managed file transfer capability between the SPE and the IQTE, as discussed in section 3.4 of this report.

The proposed development architecture appears to replace any use of the GS/SPF-based Software Test Management System with CORCASE test management.

4.3 Assessment

There are not enough details available, nor has there been sufficient time to fully investigate the CORCASE-Atherton alternative. Based on documentation from Atherton and from Verdex (the Ada compiler vendor), the tool integration aspects of the CORCASE alternative are doable. However, detailed information on CORCASE itself was considered by Loral to be proprietary information and was not available to the RICIS study team.

There are several issues to be considered from the perspective of GSDE interfaces:

1. The "batch file interface" development to integrate CORCASE with formal CM may be a significant effort, and may not be available to support early stages of software acceptance test and integration when delivery to formal CM first takes place. Without detailed CORCASE information, this risk is impossible to assess. Although the SSE CM Oracle interface definitions are published, duplicating the Rational-Oracle interface may not be trivial.
2. The relative difficulty of automating CM interfaces, a stated goal of the SSCC project, cannot be assessed.
3. The ability to bypass the GS/SPF, as CORCASE interacts directly with the test environment, raises concern that formally-controlled software may not always match the software that gets tested or even delivered.
4. STM capabilities are expected to evolve into the formal build-delivery process for Space Station software. If CORCASE is a substitute for STM, it may also have to evolve to match the SSE-developed capabilities.
5. There is no experience with CORCASE or the Atherton Software Backplane in the NASA environment. The DoD software environment is different in many respects, particularly in terms of documentation requirements and cost. CORCASE may well be an excellent solution for problems not found in the JSC environment.
6. The Atherton Software Backplane is essentially a data repository. CORCASE is essentially a data repository. It is not clear how the Atherton interfaces will interact with the CORCASE data systems.
7. The present CORCASE will have to be modified to match current requirements. It is reasonable to assume that continued modifications will be required to address changes in requirements. (The SSE project provides ample evidence of requirements evolution). Cost issues are not directly within the scope of this study, but interface and schedule and capability considerations are. The fact that such a continuing development activity, paralleling the SSE project, must occur to support the MSC alternative raises those considerations.
8. In the event that the CORCASE-Software Backplane approach is phenomenally successful, the fact that CORCASE is a proprietary tool might prevent other JSC projects from emulating that success.

The bottom line is that there are many unresolved issues involved in the SSCC project plan for CM. On the positive side, the SSCC team has consistently demonstrated a great concern for CM issues and a high degree of professionalism. They clearly take the risks of inadequate CM very seriously, and have acted to minimize those risks.

On the negative side, the SSE project started out with a proprietary software management tool, the APCE, and had a rather dramatic lack of success with it.

Section 5 - Space Station Training Facility

The Space Station Training Facility will involve workstations, standard data processors (SDPs), special-purpose input and output devices, and medium-scale mainframe computers to provide a functional simulation of Space Station Freedom. The SSTF is being developed for JSC on the Training Systems Contract. It will provide major support for mission crew training and will provide a venue for demonstrating and testing user interface systems developed for on-board use.

The SSTF software development effort will use Ada for the most part, with some special-purpose elements using C or other languages. The development environment was described in the June 1991 RICIS study report, in appendices B and D. Additional detail is provided by the "SSTF Software Configuration Management Approach" briefing, September 1991.

5.1 Overall software workflow

SSTF software developers will build software using workstations and Rational R1000 development computers. The SSE-provided STM and formal CM capabilities on the GS/SPF will be used to move software into the target environment for compilation, integration and test. The SSTF development plan incorporates a "reconfiguration Rational" that will serve as an adjunct to the reconfiguration (Recon) mainframe in the target environment. The Recon mainframe will act as the CM host for intermediate products and command scripts.

Developers will have access from workstations to the target environment in order to test interactive software, to monitor test execution, and to debug code in development. Presumably the Recon Rational will simplify the debugging process, although workstation-based text editors will also serve the purpose.

Access controls in the target environment will prevent developers from modifying resources that are used in training or are undergoing formal test. Nevertheless, the considerations applying to mixed environments, which were described in section 3 of this report, are applicable to the TSC development process.

5.2 Configuration management

The TSC CM plan describes a graduated approach to CM at different levels of rigor and authorization. This reflects a clear recognition of the operational considerations described in section 3, and is well suited to the complexity and size of the project. The TSC plan is to distribute the CM function across three different platforms based on the type of object to be managed (e.g., source, compilation products, executable loads). The stated purpose of this distributed system is to minimize waste and redundancy.

PRECEDING PAGE BLANK NOT FILMED

The TSC approach identifies the level of authority responsible for changes at each stage of development through delivery to operations and sustaining engineering. It specifically encompasses the need to have different versions in different phases of development at the same time. It states that NASA will have full visibility into the different CM systems.

5.3 Assessment

The SSTF CM plan is described as a three-part distributed integrated system. It attempts to partition the software process along lines that will minimize the need for communications across the boundaries of the distribution. Source code is in one place, compiled code in another, operational software is in a third. The intent is to store and manage products where they are used.

(The CM plan also describes a fourth element, developer-level CM, which is not addressed in this study).

The CM plan also provides a detailed map of what inputs, products, activities, and management occur at steps throughout the development life cycle. In addition, it provides a thoughtful (though negative) response to earlier RICIS study findings.

The basic assessment of the study team is that the TSC plan focuses too much on making things easy (simplifying interfaces, putting things where they'll be used, etc.), and not enough on making things secure.

Although many of the recommendations made previously have in fact been incorporated into the SSTF CM plan, there is still too little detail to determine whether or not the protections described will meet the requirements detailed in section 2 of this report.

The study team found three areas of particular concern in the SSTF CM approach: the use of automation (or lack thereof), the effectiveness of (and visibility into) distributed CM systems, and the interface complexity in distributed CM. In all three areas the major finding was a lack of information with which to make a complete assessment, coupled with an approach that emphasized development efficiency rather than configuration reliability.

CM Automation

The SSTF CM approach does not identify which elements of the process will rigorously controlled and which will be up to developer discretion. The different enforcement mechanisms are treated as interchangeable.

Citing from the September 1991 briefing:

Movement of software and data between these environments is controlled by automation, by access controls, and by procedural controls. Automated Interfaces and manual procedures will control this integration [of the distributed CM systems]. SSTF's division of these systems reduces the complexity of the interfaces.

This policy statement provides no assessable information, and ignores the reality that automated controls are markedly more effective than manual procedures *in accomplishing the goals of CM*. If programmers could be relied upon to fully comply with manual procedures, there would be no need for CM tools. Yet the SSTF outlook on CM automation is unenthusiastic (from the same briefing):

Complete automation of this interface [between development and test environments] would be expensive, would slow and complicate development, and is unfeasible at preset. TSC has learned important lessons from attempts to centralize and automate CM in previous simulation complexes.

SSTF CM process is currently as automated as is feasible.

In view of the fact that TSC has not identified what elements of the interface are automated, or what tools (COTS, SSE-provided, or developed) will be used in the CM process, the SSTF contention ("as automated as is feasible") can be neither disputed nor confirmed.

Effectiveness and visibility of distributed CM systems

This issue is quite simple: while the CM system specified for source code (the SSE CM system on the GS/SPF) is a known quantity, the Recon and Ops CM systems are unknown. If the Recon and Ops CM systems provide the same degree of rigor and visibility as the GS/SPF CM system, the risk will be small.

The emphasis of the SSE-provided formal CM system is on accountability and control, not on developer convenience. It may be painful, but will be dependable. Similarly, the degree of visibility into the SSE product is fully defined in the SSE requirements and design documentation.

Until those systems are specified and assessed, this area of risk will remain.

Distributed CM interfaces

The TSC allocation of CM functionality to different platforms for different types of products makes sense from an operational view. Source code is processed by editors which exist in the development environment, and should reside there. Compiler output is used (to build load images) in the target environment, and so on. Operationally, the allocation makes sense. However, when the requirements defined for CM are used as a basis for assessment, the effectiveness of these allocations is more ambiguous.

One requirement is to be able to reconstitute delivered software from source; another is to be able to reproduce a test session. Distributed CM complicates these activities.

To rebuild a product in the SSTF approach, source code from the GS/SPF CM system must be married with compilation scripts (and possibly intermediate libraries) from the Recon CM system. This means that if either CM system is changed, the other must be informed. (If the scripts on the Recon CM system are modified, the fact that the source code is unchanged will be all the more confusing.)

Repeating a test will involve test scripts and test data from GS/SPF CM and loads from OPS CM. If the retest involves recompiling a test article, all three CM systems become involved. All three must therefore be kept synchronized, and changes to any one must be reflected in two other places. This compounds the complexity of the interface, instead of simplifying it.

The problem with distributed CM, as TSC has designed it, is that the reliability of the system is no better than the lowest common denominator. Rather than redundancy (which would improve reliability) we have serial interfaces wherein the total process is no more secure than its least secure step.

Summary

The assessment of the study team is that TSC has put a lot of effort into formulating a comprehensive plan. This plan is substantially improved from that which was originally reviewed. TSC's approach shows clear appreciation for the technical requirements of software configuration management, and provides justifications for those areas where it deviates from the straightforward SSE-provided mechanism.

Nonetheless there is a degree of risk inherent in the lack of specific detail (e.g., the name of COTS software CM tools or the design specification of a developed tool). That risk is made more significant by what we perceive as a lack of balance between developer convenience and configuration security. Finally, we believe there is a significant likelihood that the proposed distributed CM system will fail to ensure product consistency without significant manual overhead.

Section 6 - Recommendations

The primary recommendation of this study (as it concludes) is that *some* independent review of the CM plans of both the SSCC and the SSTF projects should continue to occur. In our opinion, the responsiveness of both contractors, whether enthusiastic or grudging, has led to a greater awareness of the requirements of CM and better plans for implementing it.

A secondary recommendation is that the use of the SSE STM be strongly encouraged, and that interface tools and scripts be built to adapt STM to the ground system environment. This would serve to establish a degree of commonality between SSCC and SSTF, and would provide more uniform information to NASA on the progress of testing in the two projects.

Specific recommendations for each ground systems are discussed below.

6.1 Space Station Control Center

The basic recommendations are essentially drawn from the assessment in section 5.

1. The alternative approach presented by MSC using CORCASE as a replacement (in part) for the SSE needs to be evaluated in detail. The following specific items should be considered:
 - a. What is the commitment to maintaining and evolving CORCASE as development requirements change, and as problems arise? How can the contractual problems of the APCE (the unlamented SSE tool) be avoided for CORCASE? Who will have rights to use the modified, evolved tool?
 - b. How do two repository-based systems (CORCASE and Atherton) productively co-exist? Is information stored redundantly? Can it be shared with other systems?
 - c. What tailoring is being done to make CORCASE compatible with JSC development phases and methods?
 - d. How difficult will it be to interface CORCASE with the Oracle-based CM and STM?
 - e. What is the impact of losing the incremental-compilation capability as the R1000 compiler is replaced with the Verdix Ada compiler?
 - f. How easy is CORCASE to use in real projects? Who has experience with it and what are the positive and negative elements of that experience?
2. The CM should plan be clarified to ensure that CM storage on the GS/SPF does not become an afterthought (see figure 8). (For example, it could be required that all deliveries be generated from source code entirely from the GS/SPF.)

3. The interface with STM should be clarified and used during testing, even if CORCASE provides an alternative mechanism.

6.2 Space Station Training Facility

Most of the detailed recommendations are made in the assessment in section 5. The study team recommends:

1. The GS/SPF CM system should be used as a master CM system even if routine CM is handled by Recon or Ops CM. There should be one place which is the final authority and a complete source for regenerating products, and the formal CM system is the best tool for that job.
2. Automated procedures should be developed for generation of deliverable products.
3. System security procedures should be clarified to state that source code never goes from the integration or operational systems into the development area or into the GS/SPF. (As currently described, code being unit tested can be modified in the test environment and returned to the SPE).
4. The tools and procedures to be used for CM should be specified in detail, either as COTS products or as specific development efforts.
5. Once the tools are specified, they should be subject to independent review in terms of their configuration security and provisions for NASA visibility into the CM databases.

Glossary

AAS	Advanced Automation System: a large real-time ground support system (for air traffic management) currently under development in Ada, using Rational R1000 development computers and IBM mainframe target computers
Ada	the primary programming language for the Space Station Freedom Project. Ada is a trademark of the US Department of Defense
C	a programming language commonly used with Unix systems and applications programming
CASE	computer-aided software engineering
CM	configuration management
CORCASE	proprietary life-cycle management environment for software development, proposed for use on the MSC
COTS	commercial off-the-shelf (usually refers to software or hardware)
CR	change request: a formal request to change a requirement
CSC	Computer Sciences Corporation
DR	discrepancy report: a formal report that a system (in this case, usually software) does not meet its requirement specification
FAA	Federal Aviation Administration (customer for AAS project)
formal CM	the SSE-provided CM system residing on the GS/SPF; it manages the software that has been delivered to NASA, and provides a controlled baseline from which deliveries are made to operations
GS/SPF	Ground Systems/Software Production Facility (the mainframe computer in the GSDE that hosts the SSE-provided SPF software)
GSDE	Ground Systems Development Environment
IQTE	integration and qualification test environment (a term used in this report for the target environments of the two projects)
JSC	Lyndon B. Johnson Space Center, Houston, Texas
NASA	National Aeronautics and Space Administration
OI	operational increment (the added functionality in a new release)
QT	qualification testing--the last testing stage prior to delivery to operational use (in the GSDE life cycle)
repository	a structured collection of project data generated and used by CASE tools and maintained by a repository manager
RICIS	Research Institute for Computers and Information Systems

SDP	standard data processor (the onboard processor specified for Space Station Freedom)
snapshot	a complete stored copy of a software component or subsystem at a specified point in its development; subsequent changes to the software can be compared to the snapshot
Software BackPlane	a CASE support framework from Atherton Technology, provides repository services and tool interfaces to third-party tools
SPE	software production environment; a network of servers and workstations used for the design and development of software
SPF	software production facility: an SSE term for the combination of hardware [DEC VAX or IBM mainframe] and software [SSE-developed and COTS] that anchors a local network of development servers and workstations
SSCC	Space Station Control Center
SSE	software support environment (specifically, the SSFP SSE)
SSFP	Space Station Freedom Project
SSTF	Space Station Training Facility
STM	software test management: a capability of the SSE
STR	system (or software) trouble report: a less-formal equivalent to a DR (discrepancy report)
UHCL	University of Houston - Clear Lake